Linda Kankainen

Tietokantaratkaisut verkkosovelluksen käytettävyydessä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Mediatekniikan koulutusohjelma

Insinöörityö

10.5.2017

Metropolia

Tekijä Otsikko Sivumäärä Aika	Linda Kankainen Tietokantaratkaisut verkkosovelluksen käytettävyydessä 30 sivua 10.5.2017
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Mediatekniikka
Suuntautumisvaihtoehto	Digitaalinen media
Ohjaaja	Yliopettaja Harri Airaksinen

Insinöörityön tarkoituksena oli vertailla tietokantajärjestelmien eroavaisuuksia ja pohtia tietokannan valinnan vaikutuksia verkkosovelluksen käytettävyydessä. Vertailtavana olivat relaatiomallia noudattava tietokanta ja pilvipalvelussa toimiva NoSQL-tietokanta. Eroavaisuuksien vertailussa otettiin huomioon turvallisuuskysymykset, tiedonhaun nopeus tietokannasta sekä soveltuvuus insinöörityössä toteutettavaan projektiin.

Tietokantojen eroavaisuuksia havainnollistettiin käytännön projektin kautta, jossa toteutettiin verkkosovellus reaaliaikaiselle puoluekannatusseurannalle. Verkkosovellus perustuu käyttäjien syöttämään dataan, joka tallennetaan tietokantaan ja esitetään sivustolla reaaliaikaisella grafiikalla. Tietokannasta haetun datan visualisointiin käytettiin D3.js-nimistä Javascript-kirjastoa.

Tuloksista voidaan päätellä, ettei yhtä oikeaa vastausta ole tietokantajärjestelmän tai tietomallin valintaan. Verkkosovellukset tulee ottaa yksilöllisinä projekteina ja tehdä tietokantaan liittyvät valinnat tavoitteiden mukaan. Insinöörityön projektissa tehdyssä verkkosovelluksessa paremmaksi vaihtoehdoksi osoittautui NoSQL-tietokanta, sillä sen avulla pystytään tuomaan paremmin esille tarvittavaa reaaliaikaisuutta verrattuna relaatiotietokantaan.

Avainsanat	Tietokantaratkaisut, relaatiotietokanta, NoSQL, D3.js

Author	Linda Kankainen
Title	Database systems in web application usability
Number of Pages	30 pages
Date	10 May 2017
Degree	Bachelor of Engineering
Degree Programme	Media Technology
Specialisation option	Digital Media
Instructor	Harri Airaksinen, Principal Lecturer

The purpose of the thesis was to compare differences between database systems and consider how different database solutions effect on web application usability. There were two databases compared: a relational database that is based on relational model and a cloud-hosted NoSQL database. Safety issues, speed in information retrieving and suitability for web application development were taken into account when comparing these database systems.

The differences between database systems were demonstrated through a project, which implements a real-time web application that keeps track on peoples' interests in political parties. The web applications information is based on user submitted data, which is stored to the database. The data is shown on the website with real-time graphs visualized with D3.js JavaScript library.

The results of the study indicate that there is no one correct answer when deciding which database system to use. Web applications must be taken as unique projects and make all the decisions based on what one wants to achieve with the particular application. In this case the most suitable database for the application was the cloud-hosted NoSQL database, as it was better producing real-time data to the graphs compared to relational database.

Keywords	Database systems, relational database, NoSQL, D3.js

Sisällys

1 Johdanto

Insinöörityön tarkoituksena on tutkia erilaisia tietokantaratkaisuja ja niiden vaikutuksia verkkosovelluksen käytettävyyteen. Tutkinnan kohteina ovat perinteistä relaatiomallia noudattava tietokanta ja pilvipalvelussa toimiva reaaliaikainen NoSQL-tietokanta. Raportissa selvitetään näiden kahden tietokantamallin ominaisuuksia ja vertaillaan niiden suurimpia eroavaisuuksia. Tarkasteltavana on muun muassa, mitä eroavaisuuksia erilaisilla tietokantaratkaisuilla on kummankin tietokannan rakenteessa, tiedonkäsittelyn syntakseissa ja tietokantojen turvallisuudessa.

Tietokanta on tiedon säilytyspaikka, joka voi olla minkä kokoinen tahansa. Yksinkertaisimmillaan tietokanta voi olla vain puhelinnumero paperilla tai vastaavasti sisältää kaikki maailman puhelinnumerot. Kaikki tieto on kuitenkin tallennettu johonkin tietokantaan. On olemassa erilaisia tietokantoja, jotka eroavat esimerkiksi tiedon säilytysrakenteeltaan. Relaatiomallia noudattavan tietokannan historia on pitkä, ja se onkin ollut johtavassa asemassa jo vuosikymmenien ajan. Relaatiotietokannalle tunnusomaista on sen taulukkorakenne sekä tiedon ryhmittäminen relaatioihin. NoSQL-tietokannat taas ovat uudempia, oliomalliin perustuvia tietokantoja, jotka ovat yleistyneet vasta viime aikoina. Tietokannan rakennetta kuvataan perinteisen taulukkoskeeman sijaan puurakenteella, ja tieto voidaan tallentaa esimerkiksi avain-arvopareina.

Tietokantaratkaisuiden eroja havainnollistetaan käytännön projektin kautta. Työssä suunnitellaan ja toteutetaan verkkosovellus, jonka tarkoituksena on tallentaa käyttäjien syöttämiä tietoja ja esittää ne sivustolla helposti luettavan grafiikan avulla. Projektin perusteella analysoidaan tietokantojen eroja käytännön tasolla ja arvioidaan kummankin ratkaisun sopivuutta kyseiseen verkkosovellukseen. Kahdesta tietokantaratkaisusta valitaan sopivampi verkkosovelluksen jatkokehitykseen. Erilaisiin tietokantaratkaisuihin pohjautuvien verkkosovellusten eroavaisuuksia tutkiessa huomioidaan muun muassa tietokantaratkaisuiden hinnoittelu ja tietokantaratkaisun valinnan vaikutukset sovelluksen käytettävyydessä.

Projektin lähtökohtana on selvittää, miten pilvipalvelussa toimiva tietokantaratkaisu helpottaa sovelluskehitystä, kun uuden tietokannan rakennuksen pystyy aloittamaan hetkessä ilman suurempia asennuksia tai konfiguraatioita. Tämän seurauksena myös

yritykset pystyvät ulkoistamaan tietohallintoaan eikä enää ole tarvetta rakentaa omia fyysisiä palvelimia. Tietokantojen siirtyminen palvelumuotoon vapauttaa kuluttajia esimerkiksi tietokantojen versionhallinnasta ja tuo samalla myös uusia mahdollisuuksia, kun palveluun voidaan haluttaessa liittää lisäominaisuuksia sen sijaan, että se sisältäisi vain tietokannan.

.

2 Tietokantaratkaisut verkkosovelluksissa

2.1 Tietokannat ja niiden jaottelu

Tietokanta on paikka, johon tallennetaan tietoa. Tietokanta voi olla minkä kokoinen tahansa ja yksinkertaisimmillaan muodostua vaikka vain yhdestä sanasta paperilla. Käytännössä paperille kirjoitettu sana on kuitenkin tallennettu tietokantaan. Tämän perusteella voidaan todeta, että tietokantoja on ollut olemassa jo erittäin pitkään. Yleisesti tietokannoista puhuttaessa tarkoitetaan kuitenkin tietokoneen muistiin tallennettavaa informaatiota, jonka kehityskaaren alku voidaan sijoittaa 1900-luvun puoliväliin. (1.)

Tietokannan ja käyttäjän välisenä operaattorina toimii tietokannanhallintajärjestelmä, jonka avulla käyttäjät voivat hallita tietokantaa. Järjestelmän kautta voidaan hakea, lisätä, poistaa tai muokata tietokannan dataa kyselyillä. (2.) Tietokantamallit voidaan nykyään jakaa karkeasti kahteen eri kategoriaan: relaatiomallia noudattaviin SQL-tietokantoihin ja perinteisestä taulukkoskeemasta poikkeaviin NoSQL-tietokantoihin.

Relaatiotietokannat ovat tietokantoja, jotka hyödyntävät relaatiomallia. Relaatiomalli on lähtöisin 1970-luvulta, ja sen kehitti englantilainen tietojenkäsittelytieteen tutkija Edgar F. Codd. Relaatiomalli nousi nopeasti hallitsevaksi tietokantamalliksi syrjäyttäen aikaisemmat tietokannoissa käytössä olleet verkko- ja hierarkiamallit, ja se on edelleenkin hallitsevassa roolissa. Relaatiotietokannassa tieto esitetään tauluina, jotka koostuvat sarakkeista ja riveistä. Jokaisen taulun ensimmäisellä rivillä määritellään kunkin sarakkeen nimi, jonka alapuolelle lisätään tallennettavaa tietoa listamaisesti rivi kerrallaan. Kullekin sarakkeelle määritellään sen tietotyyppi, jota tallennettavan tiedon tulee vastata. Hyväksyttäviä tietotyyppejä ovat esimerkiksi kokonaisluvut, merkkijonot tai kellonaika. Taululle määritellään myös sen perusavain, jonka tulee olla uniikki. Mikäli tallennettavassa tiedossa ei ole valmiina arvoa, joka sopisi pääavaimeksi, sellainen voidaan myös generoida. Jokaisen tallennuksen yhteydessä voidaan tallentaa myös esimerkiksi luku, joka kasvaa aina yhdellä kokonaisluvulla, kun tietokantaan lisätään uusi rivi. (3, s. 7–10.)

Relaatiotietokannoissa olennaista on se, että taulujen välille voidaan muodostaa yhteyksiä eli relaatioita. Tämä tarkoittaa, että kaikkea tietoa ei tarvitse tallentaa useaan kertaan tietokantaan, vaan voidaan käyttää aikaisemmin tallennettua tietoa muodostamalla relaatioita taulujen välille. Yhteystyyppejä on kolmessa eri asteessa: 1:1 (yhden suhde yhteen), 1:N (yhden suhde moneen) ja M:N (monen suhde moneen). Yhden suhde yhteen -aste voi ilmetä esimerkiksi ravintolan ja sen osoitteen välillä. Ravintola voi sijaita vain yhdessä osoitteessa, ja vastaavasti yhdessä osoitteessa voi olla vain yksi ravintola. Yhden suhde moneen -astetta voidaan havainnoida esimerkiksi koulun kautta. Koulussa on useita opiskelijoita, mutta jokainen opiskelija on vain yhdessä koulussa. Monen suhde moneen -yhteys voi ilmetä opiskelijan ja kurssin välillä: opiskelija on usealla kurssilla samaan aikaan, ja kurssille osallistuu useita opiskelijoita. (4.)

Tavallisesti relaatiotietokantakokonaisuudet muodostuvat useista eri tauluista. Kun tietokanta on jaoteltu useaan osioon, data on helpommin saatavilla ja voidaan varmistua myös tiedon eheydestä. Tietokannan ei kuitenkaan tarvitse olla monen taulun kokonaisuus, jos haluttu lopputulos saadaan aikaiseksi myös yhdellä taululla noudattaen eheyssääntöjä. Tällaiseen ratkaisuun voidaan päätyä, kun halutaan tallentaa tietoa, joka ei ole yhteydessä mihinkään muuhun tietokantaan tallennettavaan tietoon – eikä sinne näin ollen pystytä muodostamaan relaatioita, jotka hyödyttäisivät tietokantaa sen toiminnallisuuden puolesta.

Tiedonhaku relaatiotietokannasta suoritetaan SQL-kyselykieltä käyttäen (Structured Query Language). Sen avulla voidaan muun muassa rajata ja järjestää hakutuloksia sekä luoda monimutkaisiakin kyselyitä useiden eri taulujen väliltä. SQL-kielen avulla voidaan myös lisätä, poistaa tai päivittää tietoa tietokantaan. (5.)

Relaatiomallia noudattavien tietokantaratkaisuiden lisäksi on olemassa myös NoSQL-tietokannat. Tällaisilla tietokannoilla tarkoitetaan suurpiirteisesti kaikkia tietokantoja, jotka eivät noudata relaatiomallia. NoSQL-tietokannoilla ei ole mitään tarkasti ennalta määriteltyä rakennetta, eikä niiden ominaisuuksia voida yleistää samalla tavalla kuin relaatiotietokantoja. On kuitenkin tiettyjä piirteitä, jotka kuuluvat NoSQL-tietokantojen yhteyteen. NoSQL-tietokannat eivät käytä SQL-kyselykieltä, vaan kyselyt suoritetaan palveluntarjoajan ohjelmointirajapinnan avulla. (6.) NoSQL-tietokannat tallentavat dataa muun muassa dokumenttimuodossa tai avain-arvopareina puurakenteeseen. Insinöörityössä käytettävän Firebase-tietokannan dokumentaatiossa opastetaan, miten tällaisen

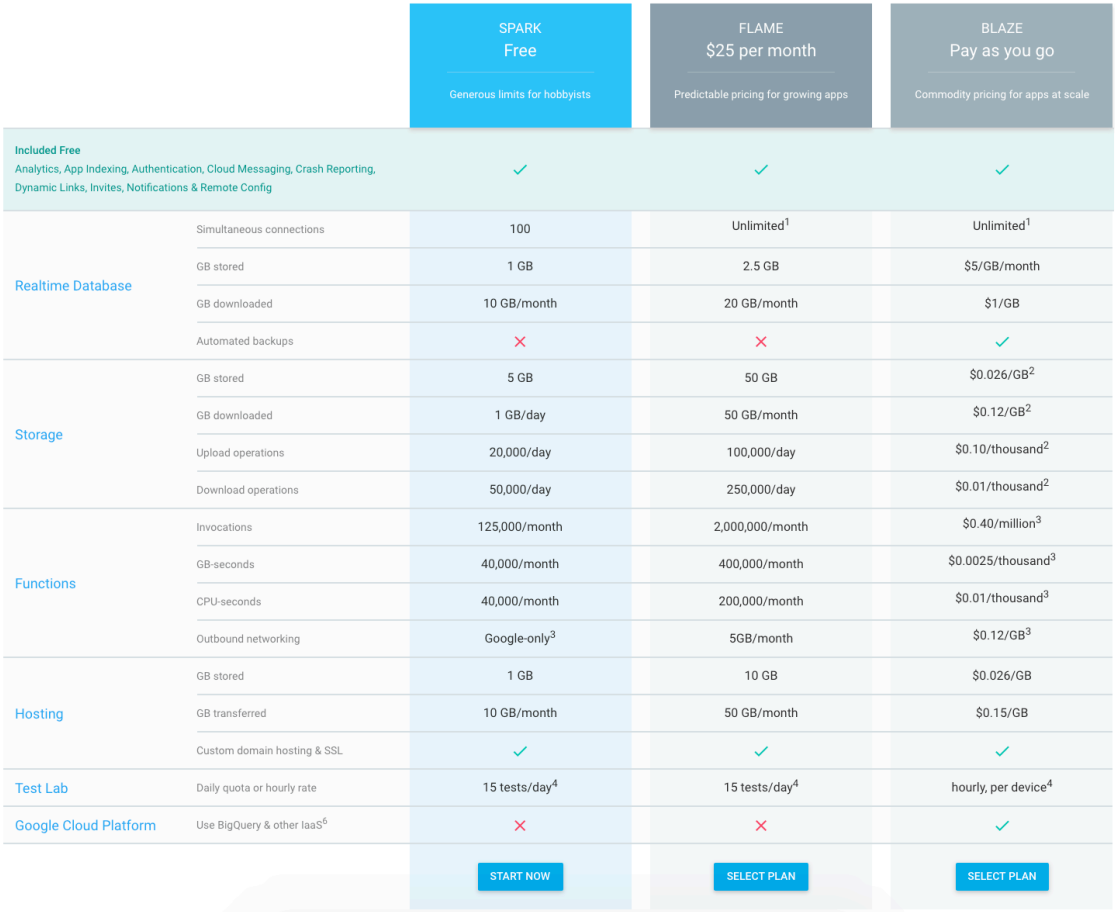
tietorakenteen tietokanta tulisi suunnitella, ja varoitetaan erityisesti liiasta datan sisäkkäin asettelusta (7).

2.2 Tietokantaratkaisuiden ominaisuudet

Verkkosovellusta toteutettaessa tärkeänä osana on tietokantaratkaisun valinta, jonka päälle sovellus rakennetaan. Sovelluksen käyttötarkoitukset ja -skenaariot täytyy spesifioida ja ennakoida mahdollisimman tarkasti, jotta voidaan nähdä kaikki hyödyt tietokantaratkaisuja ja tietomalleja vertailtaessa ja valita sen mukaan sopivin ratkaisu verkkosovellukseen.

Tietokantaratkaisuja vertailtaessa yhtenä osana on sen käyttöönoton helppous. Tietokantaratkaisun rakentamisen ei tarvitse lähteä fyysisten palvelimien asennuksesta, vaan se voidaan hankkia myös ulkoiselta taholta. Kun tietokantaratkaisu hankitaan ulkoiselta taholta puhutaan DBaaS-mallista (Database as a Service), jolloin tietokantaa tarjotaan asiakkaille pilvipalveluna. Tietokannan ylläpidosta vastaa tällöin siis palveluntarjoaja, eikä asiakkaan tarvitse itse huolehtia esimerkiksi versionhallinnasta tai varmuuskopioinnista. Tietokanta on käytettävissä heti, kun käyttäjä on rekisteröitynyt palveluun. Tämän jälkeen sovelluksen toteutus voi alkaa. Useat yritykset ovatkin jo siirtyneet palvelumuotoisiin tietokantoihin, jotka nopeuttavat ja helpottavat sovelluskehitystä. (8.)

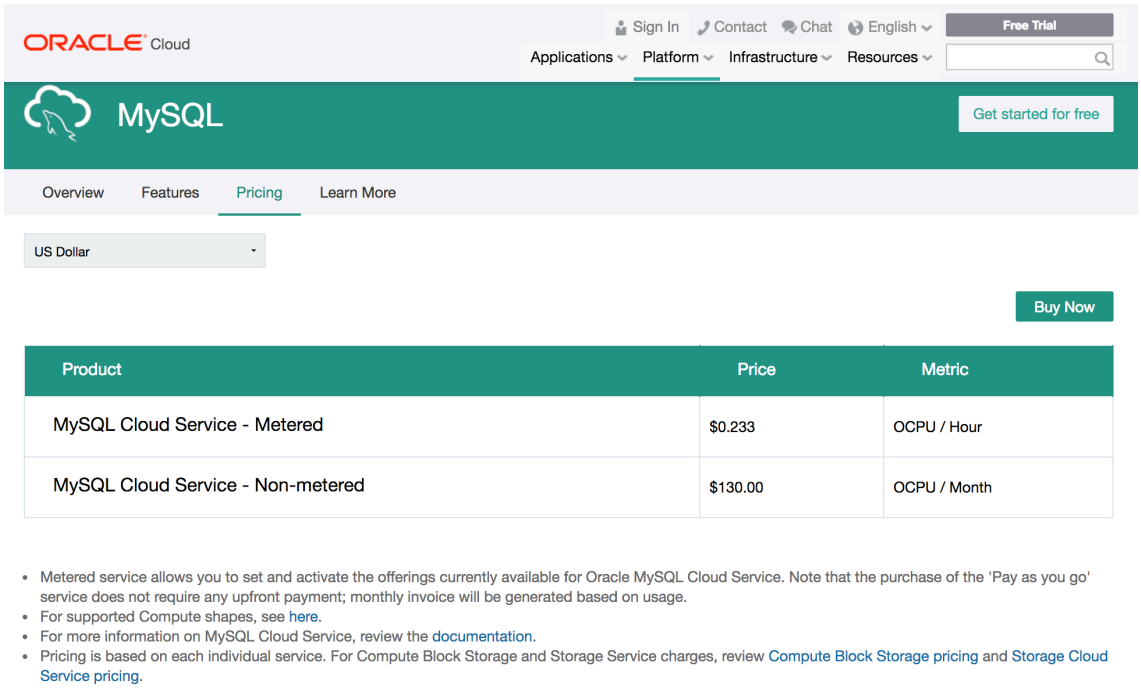
Tietokantoja vertailtaessa esiin nousee myös niiden hinnoittelu. Mikäli tietokantaa varten halutaan rakentaa oma laitteisto, sen rakentaminen voi tulla käyttäjälle kalliiksi ja aikaa vieväksi operaatioksi. Palvelumuotoisessa tietokantaratkaisussa hinnoittelu sen sijaan menee usein toteutuneen käytön mukaan. Esimerkiksi Googlen tarjoamalla Firebase-nimisellä NoSQL-tietokantapalvelulla on kolme eri hinnoitteluluokitusta (9). Vaihtoehtoina ovat ilmainen sopimus, 25 dollarin maksu joka kuukaudelta tai mahdollisuus maksaa vain toteutuvan käytön mukaan (kuva 1).



		SPARK Free Generous limits for hobbyists	FLAME $25 per month Predictable pricing for growing apps	BLAZE Pay as you go Commodity pricing for apps at scale
Included Free Analytics, App Indexing, Authentication, Cloud Messaging, Crash Reporting, Dynamic Links, Invites, Notifications & Remote Config		✓	✓	✓
Realtime Database	Simultaneous connections	100	Unlimited[1]	Unlimited[1]
	GB stored	1 GB	2.5 GB	$5/GB/month
	GB downloaded	10 GB/month	20 GB/month	$1/GB
	Automated backups	✗	✗	✓
Storage	GB stored	5 GB	50 GB	$0.026/GB[2]
	GB downloaded	1 GB/day	50 GB/month	$0.12/GB[2]
	Upload operations	20,000/day	100,000/day	$0.10/thousand[2]
	Download operations	50,000/day	250,000/day	$0.01/thousand[2]
Functions	Invocations	125,000/month	2,000,000/month	$0.40/million[3]
	GB-seconds	40,000/month	400,000/month	$0.0025/thousand[3]
	CPU-seconds	40,000/month	200,000/month	$0.01/thousand[3]
	Outbound networking	Google-only[3]	5GB/month	$0.12/GB[3]
Hosting	GB stored	1 GB	10 GB	$0.026/GB
	GB transferred	10 GB/month	50 GB/month	$0.15/GB
	Custom domain hosting & SSL	✓	✓	✓
Test Lab	Daily quota or hourly rate	15 tests/day[4]	15 tests/day[4]	hourly, per device[4]
Google Cloud Platform	Use BigQuery & other IaaS[6]	✗	✗	✓
		START NOW	SELECT PLAN	SELECT PLAN

Kuva 1. Googlen Firebase-tietokantapalvelun hinnoittelu (10).

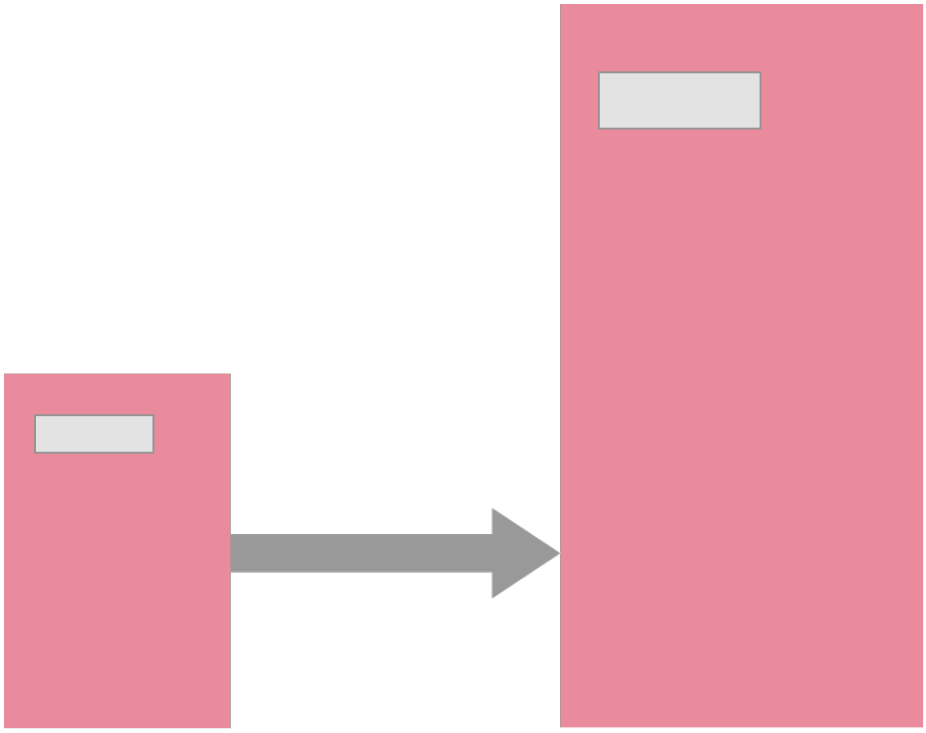
Ilmaisella jäsenyydellä on rajoituksia, mutta se sopii hyvin pienempiin sovelluksiin, jotka eivät vaadi esimerkiksi suurta tallennustilaa, yli sataa samanaikaista yhteyttä tietokantaan tai automaattista varmuuskopiointia. Tämä on myös hyvä ratkaisu sellaiselle henkilölle tai taholle, joka haluaa kokeilla järjestelmää ennen sen ostamista tai vain opetella sen toiminnallisuuksia. Myös relaatiotietokantoja on tarjolla palvelumuodossa. Kuvassa 2 voidaan nähdä samantapainen hinnoittelumalli kuin Firebase-palvelussakin: tarjolla on ilmainen kokeilu, hinnoittelu mitatun käytön mukaan tai mahdollisuus maksaa kiinteä kuukausihinta. Ilmainen versio on Oraclen MySQL-palvelussa vain 30 päivän mittainen, kun taas Firebase tarjoaa ilmaista palvelua tiettyihin käyttökriteereihin asti. (11.)



Kuva 2. Oraclen MySQL-relaatiotietokantapalvelun hinnoittelu (12).

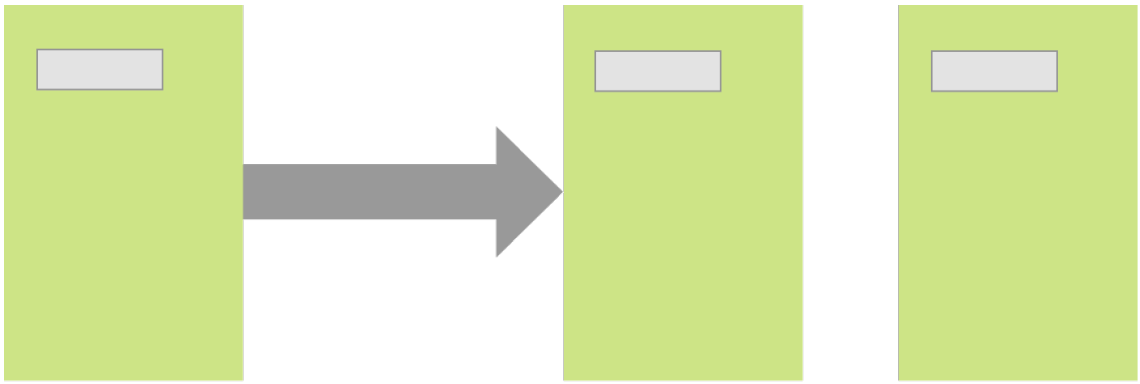
Tietokantaa suunnitellessa ja järjestelmiä vertaillessa pitää myös huomioida tietokannan skaalautumisen tarve. Skaalautuvuudella tarkoitetaan tietokantajärjestelmän koon muovautumista tarpeen mukaan tiedon määrän kasvaessa tai laskiessa. Mikäli on mahdollista, että datan määrä kasvaa erittäin suureksi nopeasti, on palvelumuotoinen tietokantaratkaisu oiva valinta uutta verkkosovellusta kehitettäessä. Jos tietokanta rakennetaan oman fyysisen laitteiston päälle, tietokannan sisällön nopea kasvu voi tulla yllätyksenä ja sen kasvattaminen tuo myös paljon lisätyötä ylläpitäjälle.

Tietokantajärjestelmät voivat skaalautua kahdella tavalla: vertikaalisesti tai horisontaalisesti. Vertikaalisella skaalautuvuudella tarkoitetaan jo olemassa olevan laitteiston komponenttien lisäämistä ja parantamista (kuva 3). Tällöin kasvatetaan jo olemassa olevaa laitteistoa eli lisätään fyysisen laitteiston tehoa. Vertikaalisella skaalautuvuudella viitataan siis siihen, että yhtä laitteistoa suurennetaan niin sanotusti ylöspäin. (13.)



Kuva 3. Vertikaalinen skaalautuvuus, jossa fyysistä laitteistoa parannetaan.

Horisontaalinen skaalautuvuus sen sijaan tarkoittaa tietokannan hajauttamista useammalle eri laitteistolle (kuva 4). Kun tietokanta on horisontaalisesti skaalautuva, voidaan lisätä fyysisten laitteiden määrää sen sijaan, että lisättäisiin vain yhden laitteiston tehoa. (13.) NoSQL-tietokannat ovat horisontaalisesti skaalautuvia, jolloin tiedon ei tarvitse sijaita samalla fyysisellä laitteistolla, kun taas relaatiotietokannat skaalautuvat vertikaalisesti. Mikäli tietokantajärjestelmä on käyttäjän omalla fyysisellä laitteistolla, tulee myös varautua sen päivittämiseen tarpeen mukaan.



Kuva 4. Horisontaalinen skaalautuvuus, jossa lisätään laitteiston määrää.

Tietokantojen ulkoistamisen yhteydessä tulee väistämättä esiin myös turvallisuuden kyseenalaistaminen. Kun tietokanta hankitaan palveluna, sen fyysistä sijaintia ei voida tietää, jolloin ei voida myöskään olla täysin varmoja sen turvallisuudesta. Kun käyttäjän ei tarvitse itse vastata tietokannan ylläpidosta, ei tiedetä, ketkä kaikki pääsevät käsiksi tallennettuun dataan sen ylläpitohenkilökunnasta. Tämä vaikuttaa myös organisaatioiden tietokantaratkaisuiden valinnassa. Vaikka lähtökohtaisesti voidaan luottaa, että suurimpien ja tunnetuimpien palvelumuotoisten tietokantajärjestelmien tarjoajat ovat turvallisia, ei kaikista arkaluontoisinta dataa uskalleta siltikään tallentaa pilvipalveluun. (14.)

3 Reaaliaikainen puoluekannatusseuranta

Insinöörityöprojektin tarkoituksena oli toteuttaa verkkosovellus, jonka avulla voidaan seurata rekisteröityjen puolueiden kannatusta. Tavoitteena oli saada äänestäjien kannatusjakaumasta reaaliaikaista dataa, jota voidaan hyödyntää esimerkiksi vaalikampanjoinnin kohdentamisessa. Käyttäjä syöttää sovellukseen oman ikänsä ja sen puolueen nimen, jota hän kannattaa. Lopullisella sivustolla näytetään vastausten jakaumaa kahden erilaisen grafiikan avulla. Ensimmäinen grafiikka kuvaa puolueiden kokonaisvaltaista kannatusta ja toisella havainnollistetaan, miten puoluekannatukset jakautuvat ikäryhmittäin.

Verkkosovelluksesta tehtiin kaksi eri versiota kahdella erilaisella tietokantaratkaisulla, joista sovellukseen paremmin soveltuva valittiin jatkokehitykseen. Tavoitteena oli tutkia, miten eri tietomallia noudattavat tietokannat eroavat toisistaan verkkosovelluksen kehityksessä ja miten nämä toteutustavat eroavat toisistaan. Sovelluksen ensimmäisessä versiossa käytettiin MySQL-tietokantaa eli perinteistä relaatiotietokantaa. Toisen version toteutuksessa sen sijaan käytettiin Googlen Firebase-palvelua, joka tarjoaa reaaliaikaista NoSQL-tietokantaa. Molempiin tietokantoihin tehtiin ensin prototyyppinä yksi grafiikka, joka näyttää kokonaisvaltaisen äänijakauman. Tämän jälkeen pohdittiin kumpaa versiota kehitetään eteenpäin esittämään myös toinen grafiikka, jossa näytetään äänijakauma ikäryhmittäin. Molempien verkkosovellusten perusrakenne toteutettiin responsiivisesti HTML5- ja CSS-ohjelmointikieliä käyttäen, ja grafiikan esittämisessä käytettiin D3.js-nimistä Javascript-kirjastoa (15).

3.1 Verkkosovelluksen ominaisuudet

Käyttöliittymää suunnitellessa täytyy minimoida lomakkeeseen mahdollisesti tehtävät virheet. Mikäli jokainen vastaaja kirjoittaisi puolueen nimen itse lomakkeeseen, tietokantoihin tulisi todennäköisesti erittäin paljon erilaisia vastauksia, mitkä taas hankaloittaisi vastausten käsittelyä grafiikkaa tehtäessä. Tämän takia lomakkeeseen ei tehty kirjoituskenttää, vaan puolueen valinta hoidetaan pudotusvalikolla. Kaikki rekisteröidyt puolueet ovat listattuna pudotusvalikkoon, ja käyttäjä valitsee niistä kannattamansa puolueen. Ikäjakauman grafiikkaa ajatellen oli kätevintä tehdä ikäryhmiä, joista vastaaja valitsee sen, johon hänen oma ikänsä kuuluu. Tällöin vältytään sekavalta grafiikalta, kun ikä näytetään ryhmittäin eikä yksittäin. Ikäryhmän valitsemiseen käytetään valintanappia, joista on mahdollista valita vain yksi ikäryhmä. Näiden lisäksi lomakkeeseen tarvittiin enää lähetys-nappi, jota painamalla annetaan käsky tallentaa tiedot tietokantaan.

Sovellukseen piti myös kehittää ominaisuus, joka estää useiden äänien syöttämisen samalta henkilöltä. Ainoa varma keino tällaisen toiminnallisuuden toteutukseen olisi jonkinlainen sisäänkirjautumisominaisuus, jossa henkilö kirjautuisi sovelluksen käyttäjäksi ulkopuolista palvelua, kuten Facebook-, Twitter- tai Google-tiliä käyttäen. Tämä luultavasti kuitenkin karsisi sovelluksen potentiaalisia käyttäjiä, sillä äänen antaminen ei olisi enää yhtä helppoa. Ennen kaikkea heräisi varmasti myös epäilys tietojen keräämisestä. Sovelluksen tavoitteena ei kuitenkaan ole kerätä tietoa, jossa voitaisiin yhdistää henkilön identiteetti hänen poliittiseen suuntautumiseensa, vaan saada yleispätevää dataa ihmisten puoluekannatuksesta eri ikäryhmissä.

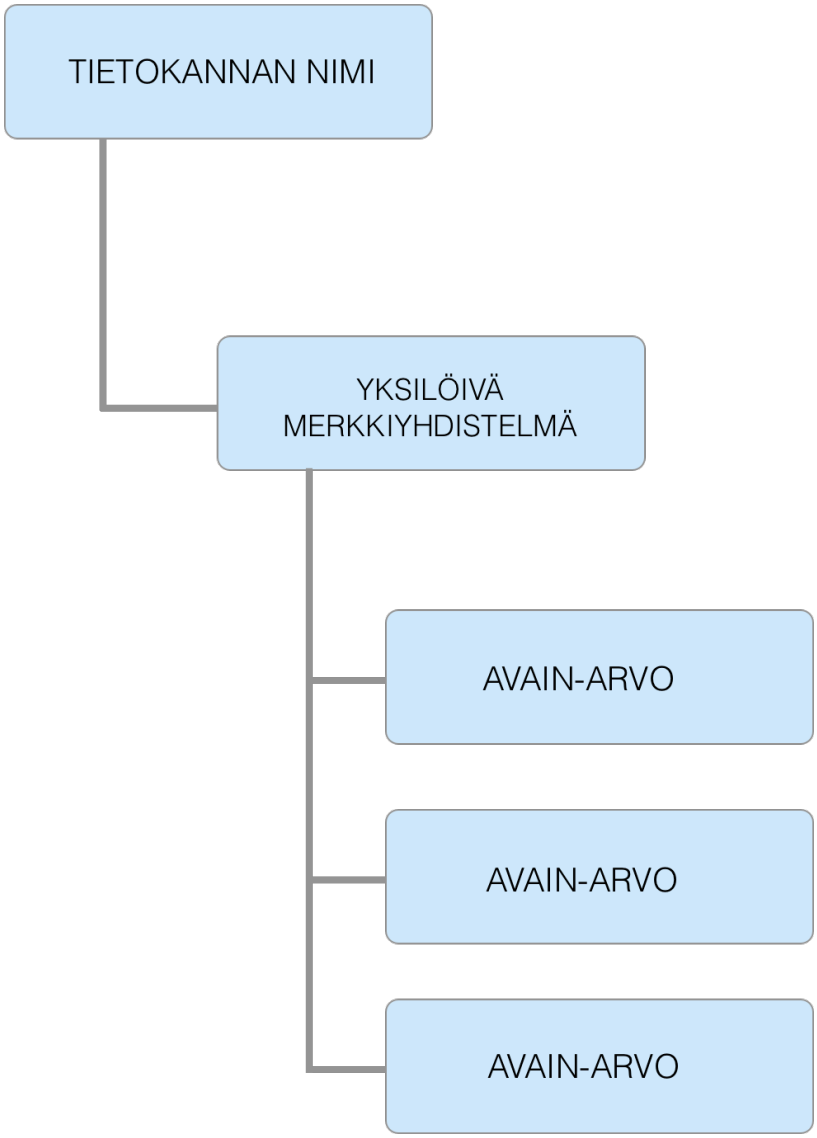
Toinen lähestymistapa useiden äänien estämiselle olisi IP-osoitteen jäljittäminen. Tällöin tarkistettaisiin, onko tietystä IP-osoitteesta jo äänestetty, ja ilmoitettaisiin sen perusteella käyttäjälle, mikäli ääni on jo annettu. Tämä olisi vielä huonompi vaihtoehto kuin sisäänkirjautuminen, sillä langattomasta verkosta pystyttäisiin äänestämään vain yhden kerran. Esimerkiksi julkisissa paikoissa olevista wifi-verkoista kyettäisiin antamaan vain yksi ääni, mikä taas ei anna ollenkaan oikeaa kuvaa kannatustilanteesta.

Sovelluksen kannalta paras tapa estää useiden äänten lähetys samalta henkilöltä toteutettiin tallentamalla käyttäjän selaimeen tieto siitä, että ääni on annettu, kun sivustolla olevaa lähetys-nappia on painettu. Arvon olemassaolo tarkistetaan ennen kuin äänestystieto tallennetaan, ja mikäli ääni on jo annettu, siitä ilmoitetaan käyttäjälle eikä

ääntä siten tallenneta. Tämä on toki melko helpostikin kierrettävissä tyhjentämällä kaikki selaimen tiedot tai avaamalla uusi selainikkuna yksityisessä tilassa. Tieto äänen tallennuksesta ei myöskään kulje selainten välillä. Siitä huolimatta se mitä luultavimmin vähentää useiden äänten antamista, ellei käyttäjä ole erikseen perehtynyt ja tietoinen tällaisista selaimen ominaisuuksista. Kääntöpuolena taas on nimenomaan se, että yhdellä selaimella pystytään normaalikäytössä antamaan ääni vain yhden kerran.

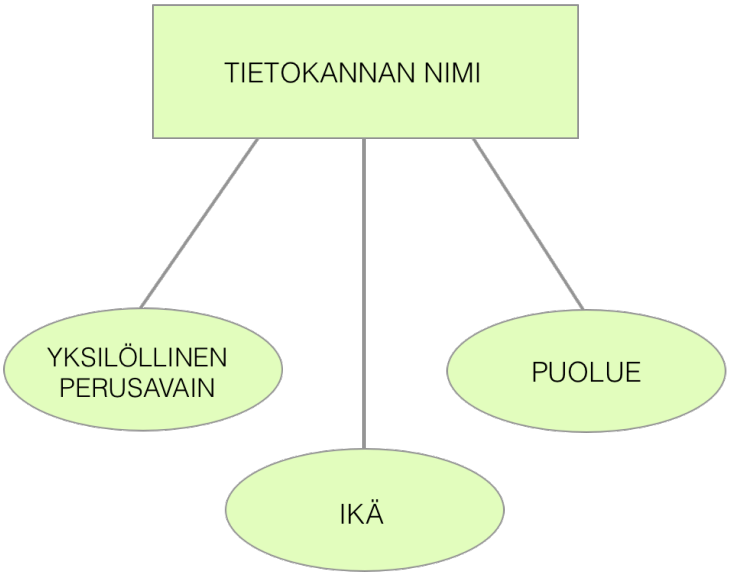
3.2 Tietokantojen mallinnus

Puoluekannatusseurannasta tehtiin kaksi eri versiota, joissa käytetään eri tietomalleja noudattavia tietokantoja. Tietokantojen suunnitteluprosessit olivat siten myös erilaisia. Ensimmäiseksi oli kuitenkin tärkeä hahmottaa, mitä tietoja halutaan tallentaa – eli ikäryhmä ja puolueen nimi. NoSQL-tietokanta ei noudata mitään tiettyä rakennetta, joten tietokantaan ei tarvinnut määritellä erikseen esimerkiksi tallennettavien tietojen tietotyyppejä tai merkkijonojen pituuksia. Firebase-palvelussa toimivaa tietokantaa suunniteltaessa päästiin alkuun pelkästään sillä, että keksittiin tietokannalle jokin nimi. Tieto tallennetaan tietokantaan avain-arvopareina, jossa avainten nimet määritellään vasta sovelluksen tallennusfunktiota tehtäessä. Tallennuksen yhteydessä Firebase generoi myös automaattisesti satunnaisen yksilöivän merkkiyhdistelmän jokaiselle uudelle tallennukselle. Verkkosovelluksen keräämä tieto tallennetaan Firebaseen kuvan 5 mukaisesti.



Kuva 5. Mallinnus Firebaseen tallennettavasta tiedosta.

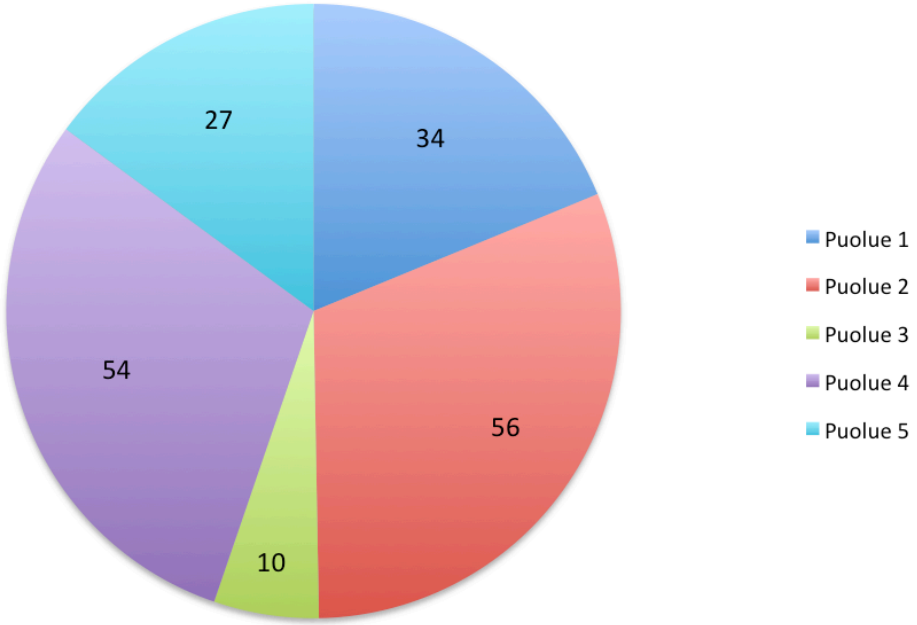
MySQL-tietokantaan tehtiin sovellusta varten vain yksi taulu, sillä sovellus ei vaadi relaatioita tai useampia tauluja toimiakseen moitteettomasti. Relaatiotietokantaa luotaessa määriteltiin taulun luonnin yhteydessä myös tietotyyppien määritykset ja sen vaatima yksilöllinen perusavain, joka kasvaa yhden kokonaisluvun verran jokaisen tallennuksen yhteydessä. Toiseen sarakkeeseen tullaan tallentamaan jokaista ikäryhmää vastaava lukuarvo ja kolmanteen sarakkeeseen puolueen nimilyhenne merkkijonona. MySQL-tietokannan mallinnus on nähtävillä kuvassa 6.



Kuva 6. MySQL-tietokannan rakenne.

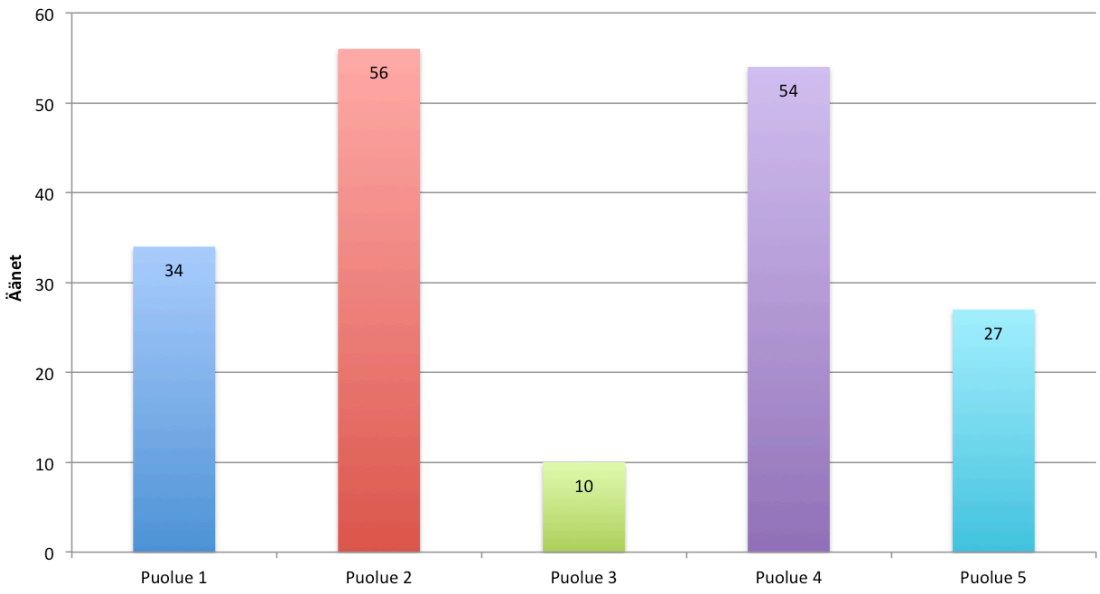
3.3 Kerätyn tiedon esittäminen

Grafiikan suunnittelussa täytyi ensimmäiseksi pohtia, mikä olisi paras muoto kerätyn tiedon esitykselle ja mitä sen halutaan kertovan sovelluksen käyttäjälle. Tulosten tulee olla selkeästi tulkittavissa ja vertailtavissa, ja grafiikasta täytyy saada nopealla silmäilyllä oikea yleisvaikutelma senhetkisestä tilastosta. Kokonaisvaltaista kannatusta tarkasteltaessa esitetään jokaisen puolueen äänimäärät erikseen, ja ikäryhmittäin jaotellussa kaaviossa tulisi ilmetä, miten nämä äänet ovat jakautuneet iän mukaan. Esimerkiksi ympyräkaavio ei sovi kummankaan tiedon esitykseen kovinkaan hyvin, sillä nopea vertailu tapahtuu käytännössä vain silmämääräisesti sektoreiden kokoja vertaamalla. Äänimäärät voivat olla hyvinkin lähellä toisiaan, jolloin tarkkojen tulosten vertailu on nopealla silmäyksellä hankalaa (kuva 7). Tilastollisessa esityksessä tavoitteena on kuitenkin antaa katsojalle tiedot mahdollisimman helposti ja vaivattomasti.



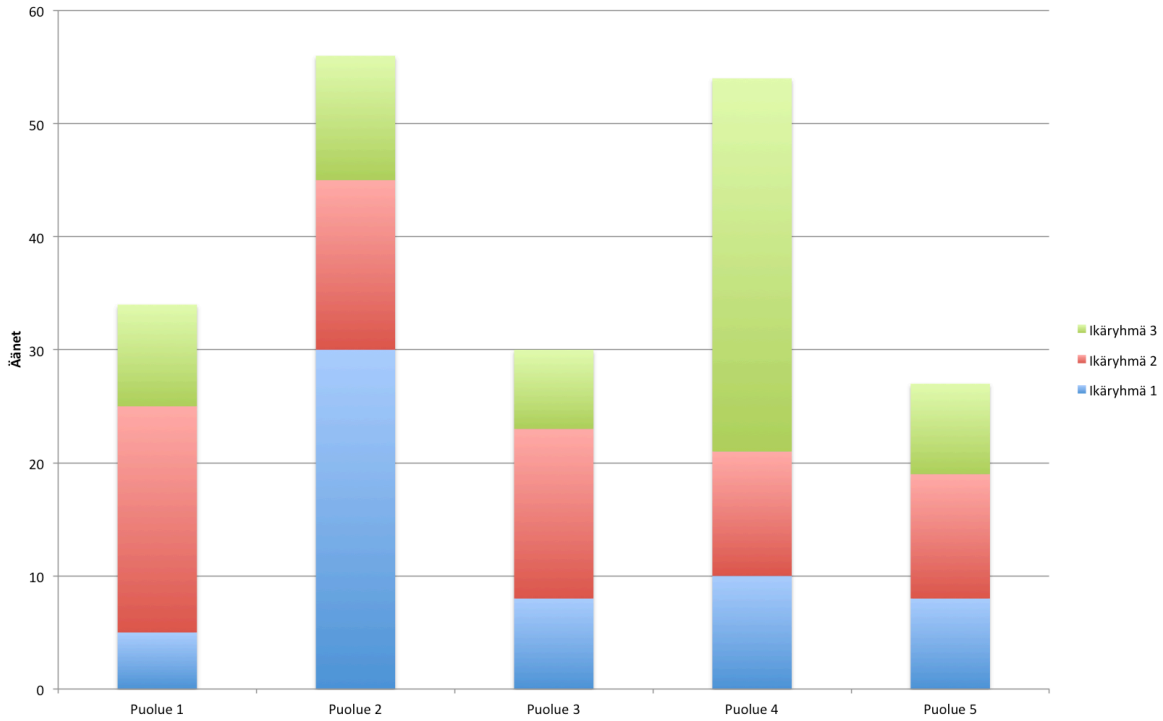
Kuva 7. Havainnollistus ympyräkaavion vaikealukuisuudesta.

Kokonaisvaltaisen puoluekannatuksen esittävään grafiikkaan valittiin pylväsdiagrammi, jonka y-akselille määriteltiin kasvavat kokonaisluvut äänille ja x-akselille jokaisen puolueen nimi. Pylväiden arvojen lukeminen on helppoa y-akselin asteikon avulla, ja pylväiden kokosuhteet ovat myös nopeasti vertailtavissa puolueiden välillä, ja ne antavat hyvän yleiskuvan äänimääristä (kuva 8). Tämän lisäksi jokaiseen pylvääseen tehtiin erillinen tekstikenttä, joka tulee esiin, kun tietokoneen kohdistinta liikutetaan pylvään kohdalla tai kun pylvästä painetaan mobiililaitteella selatessa. Tekstikentässä kerrotaan puolueelle annettujen äänien tarkka lukumäärä.



Kuva 8. Pylväsdiagrammin esitys kokonaisäänimäärästä.

Sovelluksen toisen grafiikan taas tulee esittää, miten kokonaisäänimäärät jakautuvat edelleen ikäryhmittäin. Tilaston esitykseen valittiin ositettu pylväsdiagrammi, jossa jokainen pylväs jaetaan väreihin ikäryhmien mukaan. Jokaista ikäryhmää vastaava väri kerrotaan grafiikan vieressä olevassa selosteessa. (Kuva 9.) Myös ositettuun pylväsdiagrammiin tehtiin tekstikenttä, jotta voidaan nähdä tarkat luvut eri ikäryhmien antamista äänistä jokaiselle puolueelle. Tekstikenttä koskee silloin siis jokaista ikäryhmää, eli värialuetta, josta ilmenee, kuinka monta ääntä väriä vastaavasta ikäryhmästä on tullut kyseiselle puolueelle.



Kuva 9. Ositettu pylväsdiagrammi äänten jakautumiselle ikäryhmittäin.

Värien päättäminen grafiikkaan on myös tärkeää varsinkin ikäryhmäjakauman esittävässä ositetussa pylväsdiagrammissa. Ikäryhmiä vastaavat värit eivät saa olla liian lähellä toisiaan, vaan niiden tulee selkeästi erota toisistaan. Osioissa ei voi myöskään käyttää liian vaaleaa väriä, jotta se erottuu valkoisesta taustastaan. On myös otettava huomioon, ettei diagrammin kaikkia värejä välttämättä tule näkymään jokaisessa palkissa, jos puolueelle ei tule ikäryhmältä yhtäkään ääntä. Värien välillä pitää siis olla tarpeeksi kontrastia, jotta ne erottuvat selkeästi toisistaan näkyvätpä ne sivustolla missä järjestyksessä tahansa.

4 Verkkosovelluksen toteutus erilaisilla tietokantajärjestelmillä

4.1 Tietokantojen muodostaminen

Insinöörityön projektin toteutus alkoi sovellusten tietokantojen luomisesta. Relaatiotietokanta tehtiin luvussa 3 esitetyn mallinnuksen pohjalta SQL-kyselykieltä käyttäen. Tietokannan luomisen yhteydessä määriteltiin kaikki attribuuttien ominaisuudet ja taulun

perusavain, aaniID (esimerkkikoodi 1). Taulun tekemisen jälkeen tietokanta oli valmis käytettäväksi sovelluksessa tietojen tallentamiseen.

```
CREATE TABLE aani(
          aaniID INT NOT NULL AUTO_INCEREMENT,
          ika INT NOT NULL,
          puolue VARCHAR(60) NOT NULL,
          PRIMARY KEY (aaniID)
);
```

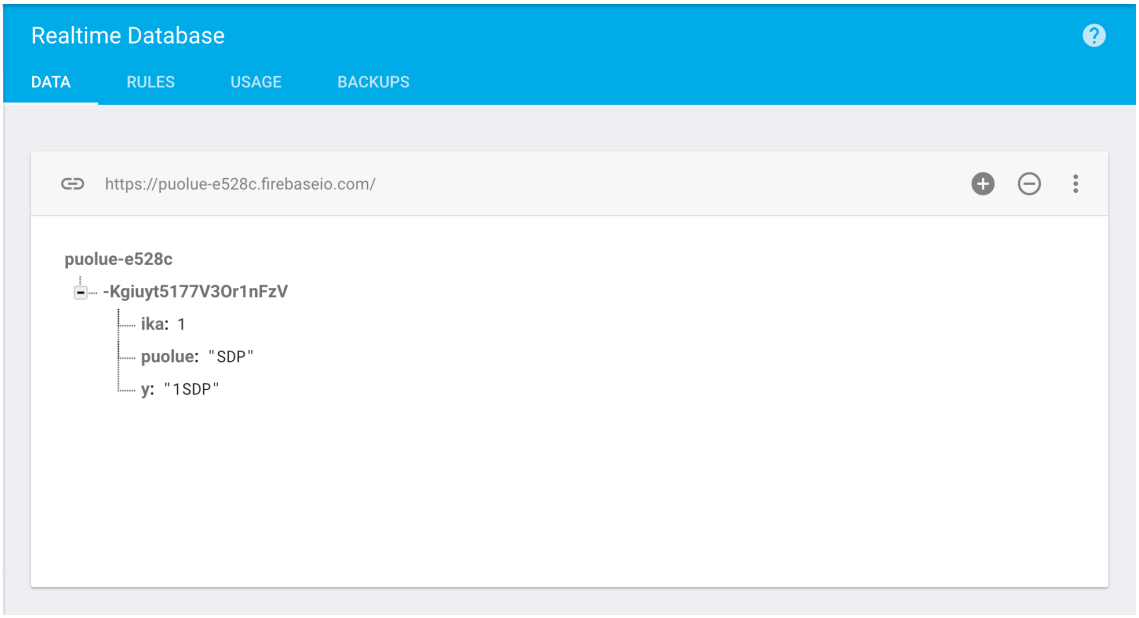
Esimerkkikoodi 1. Taulun luominen relaatiotietokantaan SQL-kyselykielen avulla.

Kyselylausekkeen ensimmäinen rivi muodostaa tietokantaan taulun nimeltä ”aani”. Tämän jälkeen tauluun määritellään sulkujen sisään sen sarakkeiden nimet, tietotyypit ja taulun perusavain. Kahden ensimmäisen sarakkeen nimiksi annetaan ”aaniID” ja ”ika”. Sarakkeiden tietotyypeiksi annetaan ”INT” (engl. integer) eli kokonaisluku. Ensimmäinen sarake tulee sisältämään uniikin perusavaimen, joka kasvaa yhden kokonaisluvun verran jokaisella tallennuksella. ”AUTO_INCREMENT”-komento hoitaa kokonaisluvun kasvamisen aina yhdellä jokaisen tallennuksen yhteydessä. Toiseen sarakkeeseen tallennetaan käyttäjän valitsema ikäryhmä. Kolmas sarake ”puolue” on tyypiltään merkkijono, joka saa sisältää 60 merkkiä. Merkkijonolle annettava maksimipituus kannattaa antaa tarpeeksi suureksi, jotta sarakkeeseen tallennettava tieto varmasti mahtuu siihen. Näiden määritysten lisäksi jokaiselle kolmelle sarakkeelle annetaan myös käsky ”NOT NULL”, mikä tarkoittaa, etteivät kyseiset kentät saa olla taulussa milloinkaan tyhjiä. Viidennellä rivillä ”aaniID”-sarakkeesta tehdään vielä perusavain ”PRIMARY KEY” -komennolla. Kuvassa 10 voidaan nähdä valmis taulu MySQL-tietokannanhallintajärjestelmässä.

	aaniID	ika	puolue
Edit Copy Delete	1	4	Vas.
Edit Copy Delete	2	6	SDP
Edit Copy Delete	3	1	RKP
Edit Copy Delete	4	2	RKP
Edit Copy Delete	5	2	EOP
Edit Copy Delete	6	1	EOP

Kuva 10. Relaatiotietokantataulu puoluekannatusseurantaa varten.

Firebasen NoSQL-tietokantaa ei tarvitse luoda kyselyillä, vaan alkuun päästään pelkällä uuden projektin luonnilla palveluun. NoSQL-tietokantaan tallennettaessa kaikki tieto tallentuu samaan puurakenteeseen, johon luodaan uusia noodeja yksilöllisellä tunnuksella. Firebase-tietokanta generoi automaattisesti tarvittavan yksilöivän nimen, joka koostuu mielivaltaisesta merkkijonosta, ellei toisin haluta määrittää. Näiden tunnusten alle tallennetaan avain-arvoparit käyttäjän syöttämästä datasta. Tallennettavat arvot ovat vastaajan syöttämä ikä ja puolueen nimi sekä näiden kahden yhdistelmästä koostuva arvo, jossa yhdistetään edelliset vastaukset. Tämän avulla äänijakauma ikäryhmittäin saadaan haettua tietokannasta helpommin grafiikkaa varten. (Kuva 11.)



Kuva 11. Firebase-tietokantaan tallennetun tiedon rakenne.

Firebase-tietokannan liittäminen verkkosovellukseen on tehty käyttäjälle hyvin helpoksi. Firebase tekee käyttäjälle Javascript-kielellä vaadittavat tiedot tietokannan alustukseen. Lyhyt koodi sisällytetään sivuston HTML-tiedoston loppuun, minkä jälkeen tietokanta on käytettävissä verkkosovelluksessa.

4.2 Verkkosovelluksen perusrakenne ja toiminnallisuudet

Kun tietokannat oli saatu käyttövalmiiksi, alkoi verkkosovelluksen toiminnallisuuksien kehittäminen. Ensimmäiseksi molemmille verkkosovelluksille luotiin samanlaiset HTML-sivut, joihin rakennettiin alkeellinen prototyyppi testaamaan käyttäjien antamien tietojen tallennusta tietokantoihin ja tietojen hakua tietokannoista. Sivulle muodostettiin lomake ja siihen yksi kenttä. Käyttäjä syöttää sivustolla olevaan kenttään numeron, joka tallennetaan tietokantaan. Numeroiden tallennus oli molemmilla toteutustavoilla suhteellisen helppoa, eikä tietojen hakeminen sivustolle tuottanut hankaluuksia. MySQL-tietokantaan tallennettujen numeroiden hakeminen tehtiin SQL-kyselylausetta ja PHP-ohjelmointikieltä käyttäen. Firebasen kanssa hakeminen pystyttiin toteuttamaan yksinkertaisella Javascript-lauseella.

Molempien tietokantojen perustoiminnallisuuksien testauksen jälkeen voitiin kehittää tarvittavia ominaisuuksia eteenpäin. Tämä aloitettiin verkkosovelluksen HTML-sivuun vaadittavien toiminnallisuuksien laajentamisesta. Lomakkeeseen tehtiin valintanappeja, joista käyttäjä voi valita sen ikäryhmän, johon hänen oma ikänsä kuuluu. Ikäryhmiä tehtiin seitsemän. Tämän jälkeen lomakkeeseen tehtiin pudotusvalikko, johon listattiin kaikki kuusitoista rekisteröityä puoluetta, joista käyttäjä valitsee haluamansa. Käyttäjän on pakko valita listasta jokin puolue, sillä oletuksena pudotusvalikossa lukee ”Valitse puolue”. Mikäli käyttäjä ei vaihda tätä kenttää joksikin puolueeksi, sivustolla kehotetaan tarkistamaan kyseinen kenttä, eikä sovellus siis tallenna mitään tietokantaan, ellei puoluetta ole erikseen valittu. Tämän tarkoituksena on estää käyttäjää lähettämästä vahingossa virheellistä ääntä tietokantaan, mikäli valikossa olisi oletuksena valmiina jonkin puolueen nimi.

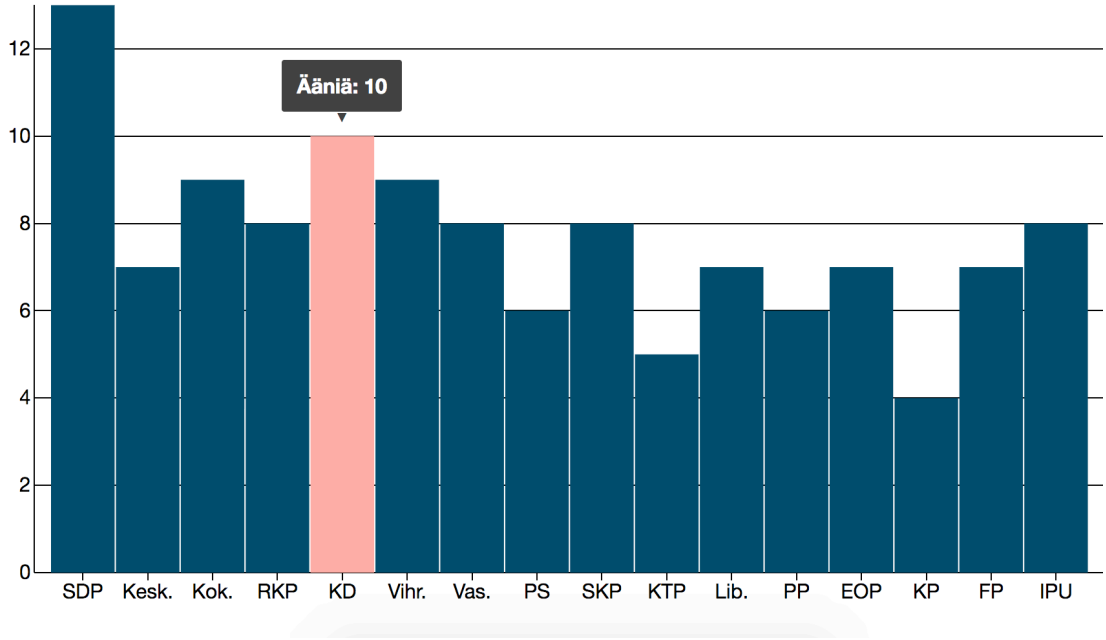
4.3 Grafiikan muodostaminen sivustolle

Seuraavana vuorossa oli grafiikan muodostaminen molemmille sivustoille käyttäjien vastauksista. Vaihtoehtoja Javascript-pohjaisille grafiikan muodostamiseen tarkoitetuille kirjastoille oli useita. Vaihtoehtoina olivat muun muassa

- Chart.js
- ZingChart
- D3.js
- Highcharts
- FusionCharts.

Grafiikan muodostamiseen valikoitui edellä mainituista D3.js-kirjasto. (15.) Kyseessä on Javascript-kirjasto, joka sisältää valmiiksi ohjelmoituja toiminnallisuuksia usean erityyppisen grafiikan piirtämiselle sovelluksiin SVG-muodossa (16). Kirjasto sisällytetään HTML-sivustolle, ja kirjasto on sen jälkeen käytettävissä koko sovelluksessa. D3.js-kirjasto valittiin toteutukseen sen lukuisten eri ominaisuuksien perusteella, ja sen avulla pystyttiin muodostamaan halutut diagrammit verkkosovelluksiin.

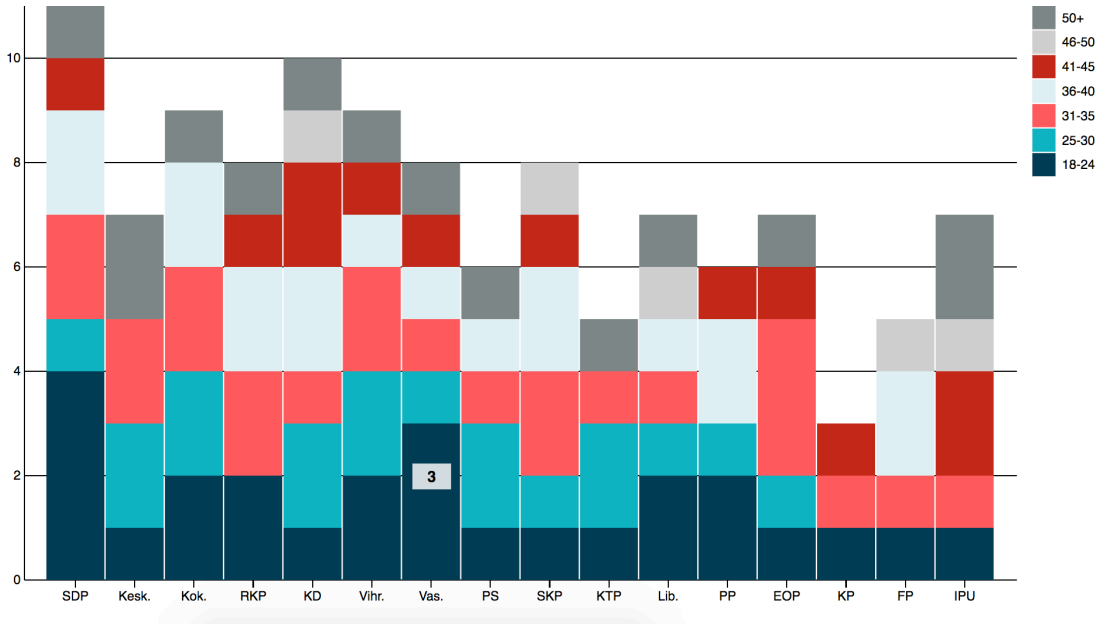
Datan haku grafiikan muodostamiseen aloitettiin tuomalla tietokantaan tallennetut objektit sivustolle ensin tekstimuodossa, jota lähdettiin muokkaamaan D3.js-kirjaston vaatimaan muotoon, jotta diagrammit saadaan piirrettyä. Ensimmäiseksi tehtiin kokonaisvaltaiset äänimäärät näyttävä grafiikka Firebasen tietokantaan tallennettujen tietojen pohjalta. Kaikki tämä hoitui Javascript-ohjelmointikieltä hyödyntäen, jota käytetään myös Firebasessa verkkosovelluksia kehitettäessä. Grafiikkaa muodostaessa haetaan kaikki tiedot tietokannasta ja esitetään pylväsdiagrammilla. Jokaisen äänen tallennuksen yhteydessä kutsutaan myös funktiota, joka lisää aina yhden äänen siihen palkkiin, jonka äänestäjä on puoluelistasta valinnut. Tämä mahdollistaa reaaliaikaisen tietojen päivityksen sivuston grafiikassa, eikä manuaalista selaimen uudelleenpäivittämistä tarvita. Diagrammin y-akselilla oleva asteikko skaalautuu automaattisesti äänimäärän kasvaessa, ja tekstikenttä näyttää aina tarkan äänimäärän jokaisen palkin yläpuolella, kun kohdistin viedään palkin päälle. Kuten kuvassa 12 voidaan havaita, kohdistimen ollessa palkin päällä sen väri muuttuu ja tekstikenttä tulee näkyviin.



Kuva 12. Valmis grafiikka puolueiden kokonaisäänimäärästä.

Ulkoasultaan samanlainen grafiikka tehtiin myös MySQL-tietokantaa käyttävään sovellukseen. Tiedon hakeminen toteutettiin eri tekniikoilla: hakemiseen käytettiin PHP-ohjelmointikieltä ja SQL-kyselykieltä, kuten aikaisemmin mainittua prototyyppiäkin tehdessä.

Kun molemmat versiot oli saatu samaan vaiheeseen, oli mahdollista vertailla käytettävyyttä erilaisten tietokantaratkaisuiden välillä. Jatkokehitykseen valikoitui Firebasea käyttävä versio sovelluksesta. Valinta tehtiin käyttötarkoituksen perusteella: tavoitteena on saada reaaliaikaista dataa, joka ilmenee käyttäjille paremmin reaalitietokantaa käytettäessä. Seuraavaksi sovellukseen tehtiin siis toinen grafiikka. Ositetun pylväsdiagrammin muodostaminen oli suhteellisen samanlainen kuin ensimmäisenkin diagrammin muodostaminen Firebaseen tallentavassa sovelluksessa. Äänimäärien hakeminen tietokannasta oli jo saatu tehtyä, joten jäljelle jäi enää niiden osittaminen ikäryhmittäin. Tekstikenttä muodostettiin diagrammiin myös eri tavalla, sillä sen tarkoitus oli näyttää vain tietyn osion äänimäärät. Tekstikenttä tulee siis näkyviin aina, kun kohdistinta liikutetaan palkin osion kohdalla näyttäen sen ikäryhmän äänimäärät, sen sijaan että näytettäisiin kokonaisäänimäärä, jonka puolue on saanut. Tämän lisäksi grafiikan oikeaan laitaan tehtiin jokaista ikäryhmää vastaavasta väristä selitykset, jotka esiintyvät taulukossa. (Kuva 13.)



Kuva 13. Ositettu pylväsdiagrammi ja äänimäärän näyttäminen yhdessä osiossa.

Grafiikan muodostamisen jälkeen täytyi enää muokata verkkosovelluksen käyttöliittymän ulkoasua ja tehdä verkkosovelluksesta responsiivinen. Toteutus hoidettiin Foundation-nimistä CSS-kirjastoa käyttäen (17). CSS-kirjastot ovat käyttötarkoitukseltaan hyvin samankaltaisia kuin Javascript-kirjastot: ne sisältävät ennalta määriteltyjä toiminnallisuuksia, joita voidaan hyödyntää oman sovelluksen kehityksessä. Foundation-kirjasto antoi hyvän kehityspohjan verkkosovelluksen ulkoasun asettelulle ja tyylittelylle. Verkkosovelluksen kehityksen viimeiseen vaiheeseen sisältyikin lähinnä värien ja tekstityylien päättäminen sekä sivuston asettelun muokkaaminen halutun näköiseksi. Grafiikka ei alkujaan ollut responsiivisessa muodossa, mutta ominaisuus toteutettiin SVG-elementeille tarkoitetun viewBox-attribuutin avulla (18).

5 Tietokantaratkaisun valinnan vaikutukset

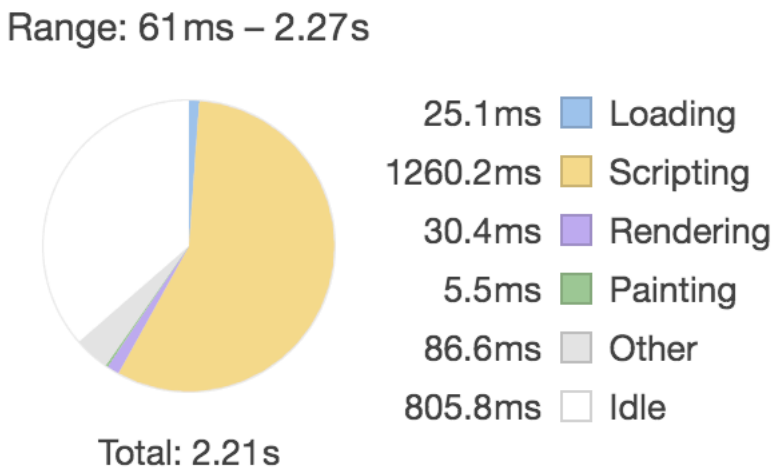
Verkkosovelluksen toteutus sujui suurelta osin helposti. MySQL-tietokannat olivat aikaisemmista projekteista tuttuja, ja ennestään tuntemattomaan Firebase-palveluun ja NoSQL-tietokantoihin löytyi suhteellisen kattavasti dokumentaatiota (19). Datan tallentaminen ja hakeminen tietokannoista ei siis tuottanut suuria hankaluuksia kumpaakaan versiota kehittäessä. Lomakkeiden teko oli myös hyvin tuttua, joten HTML-sivustolle vaadittavien toiminnallisuuksien tekeminen ei vienyt paljoa aikaa tai tuottanut ongelmia.

Grafiikan muodostamiseen käytetty D3.js-kirjasto ei myöskään ollut perusominaisuuksiltaan hankala käyttää. Sen sijaan verkkosovelluksen toteutuksessa haasteelliseksi muodostui molemmissa tapauksissa erityisesti grafiikan muodostaminen sivustoille koko ajan muuttuvasta datasta. Javascript-kirjaston yhteen sovittaminen osoittautui haasteellisimmaksi osaksi projektia, sillä erilaisten tietokantojen integroimiseen kirjaston kanssa ei löytynyt paljoakaan soveltuvaa informaatiota. Kirjaston kanssa käytetty data oli suurelta osin dokumentaatiossa tehty manuaalisesti valmiiksi esimerkiksi csv-tiedostoon (comma-separated values), eikä sitä haettu erikseen tietokannasta.

Lopputulos tietokantojen vaikutuksista verkkosovelluksen käytettävyydessä oli hyvin pitkälti odotusten mukainen. Molemmilla tietokantaratkaisuilla toteutetuilla sivustoilla saadaan näytettyä reaaliaikaista dataa käyttäjien antamista äänistä. Reaaliaikaista tietokantaa käytettäessä käyttäjän ei tarvitse erikseen päivittää verkkosivustoa, jotta uudet tulokset näkyvät grafiikassa. Grafiikka päivittyy käyttäjälle siis heti, kun tietoa on syötetty lomakkeen kautta tietokantaan – oli kyse sitten käyttäjän itse lähettämästä datasta tai muiden samaan aikaan sivustolla olevien käyttäjien antamista äänistä.

Relaatiotietokantaa käytettäessä vaaditaan erikseen verkkosivun uudelleenpäivitys, jotta käyttäjä voi huomata syöttämänsä tiedon vaikutuksen sivulla olevasta grafiikasta. Verkkosovelluksen MySQL-tietokantaan tallentavassa prototyypissä uudelleenpäivitys on toteutettu erikseen ohjelmoimalla, jolloin käyttäjän ei itse tarvitse päivittää sivua, mutta sivun uudelleenpäivitys on siitä huolimatta pakollinen. Tämä johtaa myös siihen, etteivät muiden antamat äänet kannatuksesta näy käyttäjälle, ellei hän itse päivitä aktiivisesti sivustoa. Käytettävyyden kannalta reaaliaikaisen tietokannan käyttö insinöörityön projektin kaltaisissa verkkosovelluksissa on mukavampi, sillä myös muiden tekemät lisäykset tietokantaan näkyvät jokaiselle käyttäjälle saman tien, vaikka verkkosivu olisi vain selaimessa auki. Verkkosovelluksen jatkokehitykseen valikoitui näin ollen Firebase-palvelun reaaliaikainen tietokanta.

Valmiissa verkkosovelluksessa tulee esiin kuitenkin häiritsevän pitkä latausaika, ennen kuin grafiikka näkyy sivustolla. Javascript-tiedostojen käsittelyyn menee valtaosa sivun lataukseen kuluvasta ajasta: 1260,2 millisekuntia (kuva 14).



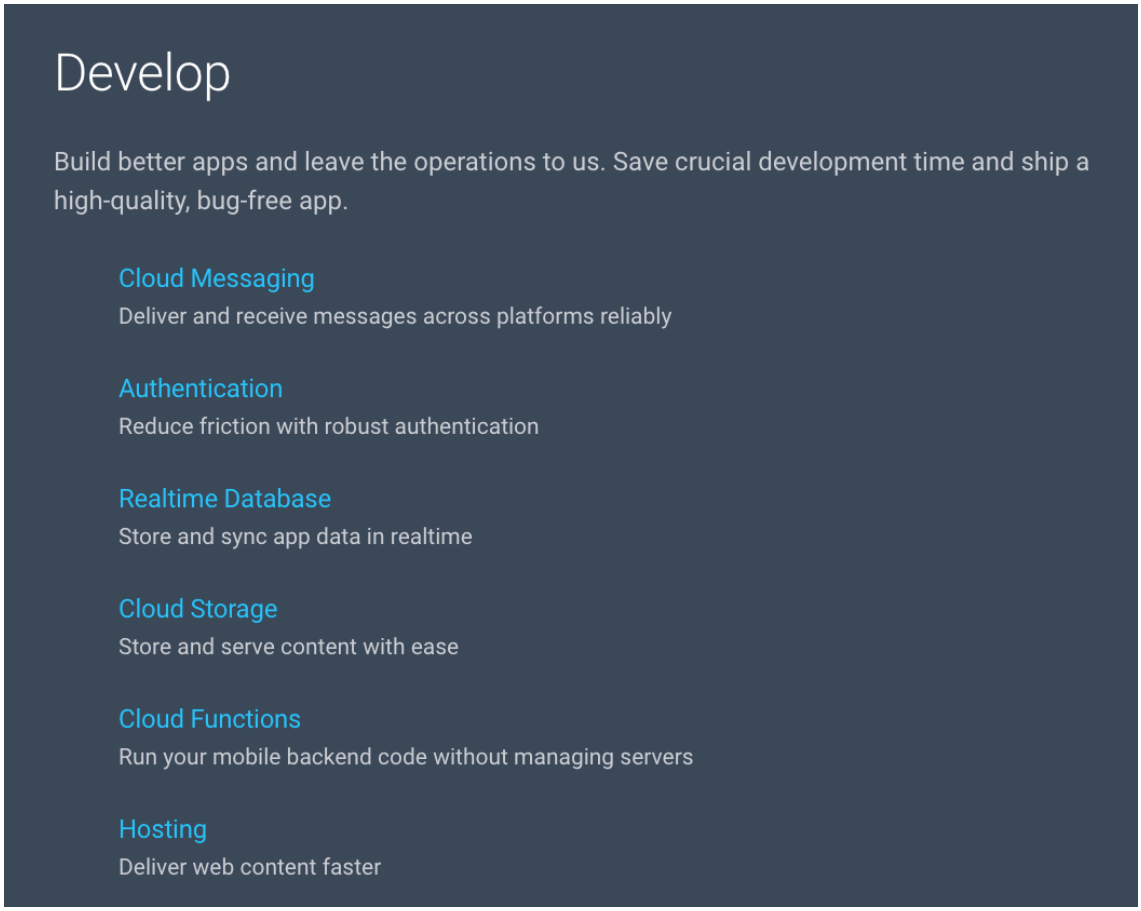
Kuva 14. Sivuston lataukseen kuluvat ajat.

Pitkä latausaika johtuu tiedonhakuprosessista Firebase-tietokannasta. Kaikki tieto haetaan verkkosovellukseen aina paikallisesti käyttäjän selaimen kautta, kun taas MySQL-tietokantaa käytettäessä tiedonhaku käsitellään palvelinpuolella, ennen kuin data ladataan käyttäjälle. Datan hakuprosessiin tarvitaan myös paljon enemmän koodia vähänkään monimutkaisempia hakuja tehtäessä Firebasesta, mikä näin ollen kasvattaa Javascript-tiedoston kokoa ja johtaa pidempään latausaikaan, ennen kuin tiedostot on suoritettu kokonaan.

Tiedon haku Javascriptin avulla aiheuttaa myös potentiaalisen turvallisuusriskin: Javascript-tiedostot näkyvät käyttäjälle selaimessa. Tämä ei itsessään näytä tietokantaan tallennettuja tietoja, vaan pelkästään tiedonhaun toteutukseen käytetyn tekniikan. MySQL-tietokantaa käytettäessä tiedonkäsittely suoritetaan palvelimella, joka ei sen sijaan anna mitään tietoja käyttäjälle selaimen kautta. Turvallisuutta ja yksityisyyttä ajatellen MySQL-tietokanta olisi täten siis parempi vaihtoehto. On kuitenkin erilaisia tekniikoita, joiden avulla Javascript-tiedostot voidaan muuttaa hankalasti luettavaan muotoon. Tällöin koodin toiminnallisuus pysyy täysin samana, mutta sen rakenne peitetään niin, etteivät muut pysty periaatteessa käyttämään tiedostoja hyödykseen. Tiedostot kannattaa myös minifioida, jolloin luettavuus vaikeutuu entisestään mutta samalla myös nopeutetaan tiedostojen suoritusaikaa. Näistäkin huolimatta kaikki Javascript-koodi tulee aina olemaan sivustolla näkyvillä käyttäjälle, ja on koodi voidaan muuttaa takaisin luettavaan muotoon. (20.)

Verkkosovellusta kehittäessä tietokannan valintaan on useita vaihtoehtoja. Tietokantoja vertaillessa täytyy pohtia, mitä sillä halutaan saavuttaa ja millainen lopputuloksen halutaan olevan. Vaikka insinöörityön projektin toteutukseen valittiinkin Firebasen reaaliaikainen tietokanta, ei se välttämättä kaikkiin verkkosovelluksiin ole oikea valinta. Mikäli dataa on paljon ja sen välillä tarvitaan useita riippuvuuksia, olisi relaatiotietokanta parempi ratkaisu tietokantoja vertailtaessa. NoSQL-tietokannat tallentavat tiedon puurakenteeseen, eikä näin ollen pystytä muodostamaan relaatioita tietojen välille. Myös kyselyiden muodostaminen Firebasesta on suhteellisen haastavaa, kun halutaan hakea paljon toisistaan riippuvaista dataa, mikä taas pitkittää sivuston latausaikaa suurien Javascript-tiedostojen takia. Perinteistä relaatiomallia noudattavassa tietokannassa tämä sen sijaan voidaan hoitaa sen relaatio-ominaisuuden avulla eikä yhtä kuormittavia kyselyitä tarvita.

Verkkosovelluksen kehityksessä voidaan myös haluta muitakin ominaisuuksia kuin pelkästään tietokantajärjestelmää sen taustalle. Firebase tarjoaa paljon muutakin kuin reaaliaikaista tietokantaa, kuten autentikaatiomenetelmän ja viestintäjärjestelmän useiden laitteiden välillä (kuva 15). Mikäli siis sovellukselle tarvitaan sisäänkirjautumisjärjestelmä, olisi Firebase sen puolesta varteenotettava vaihtoehto.



Kuva 15. Firebase-palvelun tarjoamat ominaisuudet verkkosovelluksen kehityksessä (21).

Palvelumuotoiset tietokantajärjestelmät ovat kuitenkin varteenotettavia vaihtoehtoja verkkosovelluksen kehitykseen. Tietokanta saadaan hyvin nopeasti käyttövalmiiksi, eikä käyttäjän tarvitse itse huolehtia konfiguraatioista tai varmuuskopioinnista. Tämä nopeuttaa sovelluskehitystä etenkin alkuvaiheessa ja auttaa sovelluksen ylläpidossa, kun erillisiä fyysisiä laitteistoja ei tarvita. Mikäli tietokantajärjestelmä halutaan hankkia palvelumuotoisena, jäljelle jää enää tietomallin valinta.

Projektissa käytettäviä tietokantaratkaisuja vertailtaessa on myös otettava huomioon, että Firebase on suhteellisen uusi tietokantapalvelu. Alkujaan yritys on perustettu vuonna 2011, ja Google osti sen itselleen lokakuussa 2014. Voidaankin sanoa, että palveluna Firebase on vielä kehitysvaiheessa. (22.) Tämä voi osaltaan tuoda hankaluuksia sovelluskehitykseen, vaikka yleisesti ottaen dokumentaatiota löytyykin Firebasen käyttöön kohtuullisen paljon. Päätös tietokantajärjestelmän valinnasta riippuu kuitenkin useasta eri muuttujasta ja verkkosovelluksen käyttötarkoituksesta, joten yksiselitteistä vastausta toisen tietokantajärjestelmän paremmuudesta ei ole.

6 Yhteenveto

Insinöörityössä havainnollistettiin tietokantajärjestelmien valinnan vaikutuksia verkkosovelluksen kehityksessä tietomallista riippuen ja tietokantaratkaisun eroavaisuuksia valmiissa verkkosovelluksessa. Tarkastelun kohteena olivat relaatiomallia noudattava MySQL-tietokanta ja NoSQL-tietokanta, joka puolestaan ei noudata mitään tiettyä tietomallia. Jatkokehitykseen valikoitui näistä Firebasen NoSQL-tietokanta, jonka päälle rakennettiin puolueiden kannatusta seuraava verkkosovellus.

Tavoitteena oli saada reaaliaikaista dataa käyttäjien puoluekannatuksesta ja esittää se verkkosovelluksessa kahdella erilaisella grafiikalla. Tekninen toteutus alkoi kahdella eri tietokantajärjestelmällä tehdyillä prototyypeillä, joita vertailtiin keskenään ennen jatkokehitykseen valitsemista pääasiassa verkkosovelluksen käytettävyyden näkökulmasta. Prototyypeistä saadut tulokset olivat suurelta osin ennalta arvattavissa: reaaliaikainen tietokanta näytti tiedot nimensä mukaisesti reaaliajassa, ja perinteinen tietokanta vaati sivuston latauksen uudelleen, jotta tulokset päivittyvät käyttäjälle nähtäväksi grafiikkaan.

Valmiissa verkkosovelluksessa latausaika osoittautui kuitenkin yllättäen suhteellisen pitkäksi. Kun sivusto ladataan selaimeen, kestää hetken, ennen kuin kaikki tiedot on ladattu Firebasesta ja grafiikka on valmis esitettäväksi. MySQL-tietokantaversiossa latausaika oli lyhyempi, koska tiedonkäsittely suoritetaan palvelinpuolella. Uusia tietoja lisättäessä tietokantaan verkkosovelluksen reaaliaikaisuus tulee kuitenkin paremmin esille kuin MySQL-versiota käytettäessä, vaikka ensimmäisessä latauksessa kestääkin kauemmin.

Jokainen verkkosovellus on yksilöllinen ominaisuuksiltaan ja tavoitteiltaan, jolloin myös tietokantaratkaisun valinta on sopeutettava omiin käyttötarkoituksiin. Tärkeintä on, että sovelluskehityksen alkuvaiheessa pystytään kartoittamaan, millaisia ominaisuuksia lopputuotokselta halutaan. Jos tarvitaan esimerkiksi useita muitakin ominaisuuksia kuin tietokanta, voi Firebase olla hyvä ratkaisu siihen kuuluvien palveluiden takia. Osassa projekteista taas on parempi käyttää relaatiotietokantaa, jolloin kaikki raskaampi tiedonkäsittely voidaan hoitaa palvelinpuolella eikä tarvitse kuormittaa käyttäjän sovelluksia ja laitteistoa. Myös turvallisuuden näkökulmasta tämä on mielestäni parempi vaihtoehto, sillä Firebase-tietokannassa tiedonhakuun käytettävä Javascript-ohjelmointikieli näkyy käyttäjälle selaimessa, toisin kuin MySQL-tietokannoissa, joissa käytetään pää-

asiassa PHP-ohjelmointikieltä tiedonkäsittelyssä. Projektin pohjalta voidaan todeta, ettei ole yhtä oikeaa vastausta tietokantajärjestelmän valitsemiseen.

Lähteet

1 Fortune, Stephen. 2014. A Brief History of Databases. Verkkodokumentti. <http://avant.org/project/history-of-databases/>. 27.2.2014. Luettu 10.12.2016.

2 Chapple, Mike. 2016. Database Management System. Verkkodokumentti. <https://www.thoughtco.com/database-management-system-1019609>. Päivitetty 20.11.2016. Luettu 1.12.2016.

3 Hovi, A., Huotari, J. & Lahdenmäki, T. 2005. Tietokantojen suunnittelu ja indeksointi. Porvoo: Docendo.

4 Harkins, Susan. 2003. Relational databases: Defining relationships between database tables. Verkkodokumentti. <http://www.techrepublic.com/article/relational-databases-defining-relationships-between-database-tables/>. 30.3.2003. Luettu 5.1.2017.

5 Structured Query Language (SQL). 2017. Verkkodokumentti. Microsoft. <https://docs.microsoft.com/en-us/sql/odbc/reference/structured-query-language-sql>. 19.1.2017. Luettu 3.2.2017.

6 Konkka, Petri. 2016. Tietokannat: NoSQL ja MongoDB. Verkkodokumentti. <https://petrikonkka.com/fi/pilvipalvelujen-tietokannat-nosql-mongodb/>. 28.3.2016. Luettu 9.1.2017.

7 Structure Your Database. 2017. Verkkodokumentti. Firebase. <https://firebase.google.com/docs/database/web/structure-data>. Päivitetty 3.4.2017. Luettu 4.4.2017.

8 Hovi, Ari. 2016. Tietokanta palveluna yleistyy. Verkkodokumentti. <http://www.tivi.fi/Kaikki_uutiset/tietokanta-palveluna-yleistyy-6592415>. 20.10.2016. Luettu 25.2.2017.

9 Firebase. Verkkodokumentti. Firebase. <https://firebase.google.com/>. Luettu 11.11.2016.

10 Firebase Pricing. Verkkodokumentti. Firebase. <https://firebase.google.com/pricing/>. Luettu 5.3.2017.

11 Experience Oracle Cloud with US$300 in free credits. Verkkodokumentti. Oracle Cloud. <https://shop.oracle.com/apex/f?p=cloud:free&intcmp=cloud-tryit>. Luettu 4.3.2017.

12 Oracle Cloud MySQL Pricing. Verkkodokumentti. Oracle Cloud. <https://cloud.oracle.com/en_US/mysql/pricing>. Luettu 5.3.2017.

13 Beaumont, David. 2014. How to explain vertical and horizontal scaling in the cloud. Verkkodokumentti. <https://www.ibm.com/blogs/cloud-computing/2014/04/explain-vertical-horizontal-scaling-cloud/>. 9.4.2014. Luettu 27.2.2017.

14 Francis, Ryan. 2016. Top 10 DBaaS security concerns. Verkkodokumentti. <http://www.csoonline.com/article/3049182/data-protection/top-10-dbaas-security-concerns.html>. 30.3.2016. Luettu 6.4.2017.

15 Suda, Brian & Hampton-Smith, Sam. 2017. The 38 best tools for data visualization. Verkkodokumentti. <http://www.creativebloq.com/design-tools/data-visualization-712402>. 7.2.2017. Luettu 6.4.2017.

16 Bostock, Mike. 2015. Data-Driven Documents. Verkkodokumentti. <https://d3js.org/>. 2015. Luettu 17.1.2017.

17 Foundation. Verkkodokumentti. Zurb Foundation. <http://foundation.zurb.com/>. Luettu 3.4.2017.

18 ViewBox. Verkkodokumentti. Mozilla Developer Network. <https://developer.mozilla.org/en/docs/Web/SVG/Attribute/viewBox>. Päivitetty 19.8.2016. Luettu 3.4.2017.

19 Firebase Documentation. Verkkodokumentti. Firebase. <https://firebase.google.com/docs/web/setup>. Luettu 15.12.2016.

20 Javascript Obfuscator. Verkkodokumentti. Javascript Obfuscator. <http://www.javascriptobfuscator.com/Default.aspx>. Luettu 7.4.2017.

21 Firebase Features. Verkkodokumentti. Firebase. <https://firebase.google.com/features/>. Luettu 9.11.2016.

22 Firebase. Verkkodokumentti. Crunchbase. <https://www.crunchbase.com/organization/firebase#/entity>. Luettu 1.4.2017.